Formalized Mathematics, Proof Animation, and Limit Computable Mathematics (Relevance and Feasibility of Mathematical Analysis on the Computer)

AUTHOR(S):

Hayashi, Susumu

CITATION:

Hayashi, Susumu. Formalized Mathematics, Proof Animation, and Limit Computable Mathematics (Relevance and Feasibility of Mathematical Analysis on the Computer). 数理解析研究所講究録 2000, 1169: 102-108

ISSUE DATE:

2000-09

URL:

http://hdl.handle.net/2433/64393

数理解析研究所講究録
1169巻 2000年 102-108

Formalized Mathematics, Proof Animation, and Limit Computable Mathematics

Susumu Hayashi*
Department of Computer and Systems Engineering,
Faculty of Engineering, Kobe University,
1-1 Rokko-dai, Nada, Kobe, Japan

July 19, 2000

1 Formal Proof Developments

Formalized mathematics or formal proof developments are activities to build formal proofs checked on computers for applications to formal methods or its own sake. Formal proof developments by proof-checkers, HOL, Coq, Mizar, etc are becoming realistic thanks to accumulation of proof libraries and with help of some high-tech efforts.

Bugs in softwares for some safety critical systems, such as traffic control systems, medical devices, nuclear power stations, are very risky. Even bugs in ordinary commercial softwares and hardwares can cost much for users and firms. Thus, *formal methods*, which is a technology for verifying correctness of such systems, are used. Formal methods use some kind of formal logics to verify these systems. Since human beings make errors, formal proofs on papers may be incorrect. Thus, it is recommended to verify via formal proofs on computers.

Formal methods are comparable with the activity of traditional applied mathematics. In applied mathematics, real worlds are "formalized" by dif-

*Supported by No. 10480063, Monbusyo, Kaken-hi (the aid of Scientific Research, The Ministry of Education)

ferential equations as formal methods describe the world by logical formulas of formal logics. Verifying softwares are comparable with solving differential equations.

2 Proof Animation

Developments of formal proofs are costly and heavy tasks. We may compare developments of formal proofs with finding exact solutions of differential equations. Mathematicians solve differential equations numerically and guess exact solutions by observing numerical solutions. This way of research is often easier than guessing exact solutions by bare hands.

We have proposed *proof animation* [4], a methodology for proof developments, which is analogous to this activity. We "execute" proofs under developments to find bugs in proofs just as programmers execute programs under developments to find bugs in programs. This technology is expected to make formal proof developments much easier.

"Executing proofs" means to follow proofs on examples as every mathematician does. Such an execution may be automatically done when proofs are represented as data on computers. Actually, it is known that execution of proofs are possible, when proofs are constructive. A proof is constructive, when *the law of excluded middle* is not used in the proof. The law of excluded middle is the logical principle that every proposition is true or false: $A \vee \neg A$ in symbol. There are some proof checkers which execute proofs by the principle of "Curry-Howard" isomorphism or related concepts, e.g. realizability interpretations [5].

However, these proof checkers may not be very useful for proof animations, since majority of proofs are non-constructive. In mathematics, classical logic is freely used. Restricting logic to constructive fragments are not very natural form the standpoint of ordinary mathematics. For example, it is not allowed to say that 0123456789 appears in the expansion of π or not. Existence proofs by contradiction is basically non-constructive. Thus, even if you can prove that it is contradictory that 0123456789 does not appear in the expansion of π, you cannot conclude that 0123456789 appears.

Many mathematical propositions in computer science are constructive. However, there are some important propositions on concurrent processing and combinatorics for which non-constructive arguments are much more nat-

ural.

In the late 80's, an extension of Curry-Howard isomorphism was found by Tim Griffin. He extended Curry-Howard isomorphism to classical logic and programs with continuation, which is a control mechanism used in functional programming languages.

After his work, number of theories aiming to bridge classical proofs and programs have been proposed. However, none of them looked fine for proof animation. Curry-Howard isomorphism is a rather straightforward correspondence. It is not difficult to predict outlines of programs associated to formal proofs. By such a correspondence, we can locate bugs in proofs from the location of bugs of programs associated.

However, the new Curry-Howard isomorphisms for classical logic did not provide such correspondences. The programs associated to standard proofs and/or their behavior are extremely complicated and difficult to understand. Since proof animation is a mean to *understand* proofs via observations of the behaviors of programs associated, these theories are not good enough for proof animation.

It should be noted that these theories can be practical for other aims. For example, some implementations of $\lambda\mu$-calculus, which is one of such calculi associated to classical logic, show that it is good for a kernel of functional languages with continuation.

Although, these works look hopeless for proof animation, Berardi's theory of approximation theory was an exception. Unlike to the other theories, it was aimed to understand computational contents of actual proofs. By his theory, Berardi gave an interesting computational meaning to a classical proof of the following theorem:

$$\forall f \in Nat \to Nat. \exists n \in . \forall x \in Nat. f(n) \leq f(x)$$

The "algorithm" which Berardi associated was an algorithmic process "guessing" the solution. It is described as follows:

> Regard the function f as a stream $f(1), f(2), f(3), \cdots$. Have a box of a *Nat*number. Put $f(1)$ in the box. Compare the content of the box with the next element of the stream. If the new one is smaller than the number in your box, put the new one in the box. Repeat it infinitely.

In what sense the algorithm compute the answer? The process does not stop and we will never know when we should stop. However, the box will eventually contain the correct answer. Once the box contains the correct answer, then the content will never been changed. In this sense, this non-terminating process computes the right answer *in the limit.*

This kind of "limit process" is used as a model of *learning* by Gold [3] and Putnum [?]. The value v of a limit $\lim_t f(t)$ is defined by the condition "there is N such that $f(N+i) = v$ for all $i \in Nat$. If $f(x) = \lim_t g(x,t)$ holds for a recursive function g, then f is called *limiting recursive.* We may think the sequence $g(x,1), g(x,2), \cdots$ is a trace of guessing the value of $f(x)$ on a discrete time $t = 1, 2, \cdots$.

Berardi's interpretation uses infinitary proof figures and higher order functional etc. and so was not really intuitive. He extracted the algorithm above from a standard classical proof of the proposition. However, he did not give an algorithm by which one can extract the algorithm explained above. The algorithm was obtained by an analysis of his interpretation and was not given directly from the interpretation.

3 Limit Computable Mathematics

We have found that Berardi's idea of utilizing limiting function as a kind of execution of classical proofs enables direct interpretation of a fragment of classical logic. The fragment is obtained by restricting the law of excluded middle (LEM) to Σ^0_1-formulas.

This restricted form of LEM, Σ^0_1-LEM, coincides with E. Bishop's "the limited principle of omni-science." It is essentially the algorithm explained above, i.e. Σ^0_1-LEM maintains that we may know if n $f(n) = 0$ holds for all n or there is a counterexample for $f(n) = 0$. For example, we may conclude 123456789123456789 appears in the decimal expansion of π or not by Σ^0_1-LEM.

E. Bishop wrote in his monograph of constructive analysis [2], if Σ^0_1-LEM (the limited principle of omniscience in his terminology) is allowed, "theorem after theorem of classical mathematics" can be proved. The examples he gave were the ergodic theorem, the Hahn-Banach theorem, the fixed-point theorems, etc. We also found that D. Hilbert's early work on invariant theory is also proved by the principle.

Thus, we conjecture that

> Limit Computable Mathematics (LCM) would cover a large part of practical classical mathematics, e.g., mathematics for theoretical computer science (theory of algorithms), applied mathematics, and mathematics before pure abstract mathematics of 20th century.

From theoretical point of view, extracting "semi-algorithms" from constructive proofs by Σ^0_1-LEM is fairy simple. We simply replace recursive functions in Kleene's realizability interpretation with limit-recursive functions. Then everything works just as it was. This is because of "limiting computable calculus" is also a "computable calculus". For example, if a "limit" of computational structure ω-BRFT is again ω-BRFT. ω-BRFT is known as one of the most general framework for recursion theory [7].

4 Hilbert's invariant theory: a target for case study

There are several interesting targets of case studies of proof animation by LCM. From computer science and related finite mathematics, concurrent algorithms, e.g. Dekker algorithm, and some combinatorial principles, e.g. Higman's lemma, look interesting. From works by Herbelin and Coquand, we suspect that ordinary classical proof of Higman's lemma might be beyond of feasible LCM.

From mathematics, Hilbert's invariant theory in the late 19th century seems most interesting. In his 1890 paper, Hilbert proved a theorem called "finite basis theorem". (He called it "general finiteness theorem".) In his formulation, it read that

> If $F_1, F_2, \cdots$ is a stream (sequence) of forms (homogeneous polynomials) with a fixed number of variables, then there is m such that every F_i is denoted as $A_1F_1 + A_2F_2 + \cdots + A_mF_m$ by some forms $A_1, A_2, \cdots, A_m$.

Note that m is limiting recursive in the stream $F_1, F_2, \cdots$, as far as the theorem holds. We can just "search" such m by try-and-error process. (If

we find there is F_i which cannot be represented in the form, we increment m so that $m \geq i$. This can be seen as an animation of the *statement* of the theorem. However, this is not the thing we should do.

We are planning to animate Hilbert's *proof* rather than the statement of the theorem. We have analyzed Hilbert's proof and found that the proof uses only Σ^0_1-LEM. This may be interesting, since the proof is known as one of the first transcendental proofs in algebra. Hilbert solved the long standing "Gordan's problem" once and for all by this simple lemma. Then Gordan is said to reply that "It's not a mathematics but a theology" [9].

Hilbert proved the theorem by induction on the number of variables. For the base case, the stream $F_1, F_2, \cdots$ is $c_1 x^{i_1}, c_2 x^{i_2}, \cdots$.Hilbert pointed out that the "basis" is given by a single formula $c_m x^{i_m}$, where i_m is the smallest number of the degree of the forms $i_1, i_2, \cdots$. This is the same algorithm as Berardi's.

After giving the proof of the base case, he gave an interesting specific proof for the case of two variables, and then proved the general induction step again by Σ^0_1-LEM. These proofs for the cases of two or more variables contain several applications of Σ^0_1-LEM. It is not very difficult to read its computable contents *intuitively* from the proofs. Each instance of Σ^0_1-LEM generates its own guessing sequence of a series of algebraic forms or so. Since our realizability interpretation merge all "local time" in a global time, the extracted term by the interpretation might not be good to understand the computable content. The algorithm read intuitively from Hilbert's proof suggest that there are some complicated interactions between these instances of Σ^0_1-LEM. Thus, for "legible" proof animaiton, the algorithm would have to be presented in a network of some basic guessing functions corresponding to Σ^0_1-LEM.

We are now planning to formalize Hilbert's invariant theory including his finite basis theorem via Coq proof checker and extract limiting algorithms from its proofs.

References

[1] S. Baratella and S. Berardi, Constructivization via Approximations and Examples, *Theories of Types and Proofs*, M. Takahashi, M. Okada and M. Dezani-Ciancaglini eds., MSJ Memories Vol. 2, pp.177-205.

[2] E. Bishop, *Foundations of Constructive Mathematics*, McGraw-Hill, 1970.

[3] E. M. Gold, Limiting Recursion, *The Journal of Symbolic Logic*, 30 (1965), pp.28-48.

[4] S. Hayashi, R. Sumitomo and K. Shii, Towards Animation of Proofs - testing proofs by examples -, *Theoretical Computer Science*, to appear.

[5] S. Hayashi and H. Nakano, *PX: A Computational Logic*, (The MIT Press, 1988)

[6] S. C. Kleene, On the interpretation of intuitionistic number theory, *The Journal of Symbolic Logic*, 10 (1945), pp.109-124.

[7] P. G. Odifreddi, *Classical Recursion Theory*, North-Holland.

[8] H. Putnam, Trial and Error Predicates and the Solution to a Problem of Mostowski, *The Journal of Symbolic Logic*, 30 (1965), pp.49-57.

[9] C. Reid, *Hilbert*, Springer-Verlag, New York, 1996

[10] A. S. Troelstra and D. van Dalen, *Constructivism In Mathematics*, Vol. I and II, North-Holland.